World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:12, 2009

A Growing Natural Gas Approach for Evaluating Quality of Software Modules

Parvinder S. Sandhu, Sandeep Khimta, Kiranpreet Kaur

***Abstract*—**The prediction of Software quality during development life cycle of software project helps the development organization to make efficient use of available resource to produce the product of highest quality. "Whether a module is faulty or not" approach can be used to predict quality of a software module. There are numbers of software quality prediction models described in the literature based upon genetic algorithms, artificial neural network and other data mining algorithms. One of the promising aspects for quality prediction is based on clustering techniques. Most quality prediction models that are based on clustering techniques make use of K-means, Mixture-of-Guassians, Self-Organizing Map, Neural Gas and fuzzy K-means algorithm for prediction. In all these techniques a predefined structure is required that is number of neurons or clusters should be known before we start clustering process. But in case of Growing Neural Gas there is no need of pre-determining the quantity of neurons and the topology of the structure to be used and it starts with a minimal neurons structure that is incremented during training until it reaches a maximum number user defined limits for clusters. Hence, in this work we have used Growing Neural Gas as underlying cluster algorithm that produces the initial set of labeled cluster from training data set and thereafter this set of clusters is used to predict the quality of test data set of software modules. The best testing results shows 80% accuracy in evaluating the quality of software modules. Hence, the proposed technique can be used by programmers in evaluating the quality of modules during software development.

***Keywords*—**Growing Neural Gas, data clustering, fault prediction.

I. Introduction

SOFTWARE quality assurance is an important part of any software project. In order to appraise the quality of any software product we make use of Software quality estimation models. These quality models can be used to identify program modules that are likely to be defected [1] [2]. It helps project manager to make efficient use of limited resources to target those modules that are defected [3]. A software quality models help development team to track and detect potential software defects during development cycle and saving lots of efforts that are later required for the maintenance of that product. A software quality model is trained using software measurement and defect data of a previously developed release or similar project [4], [5]. The trained model is then applied to modules of the current project to estimate their quality. The proposed work is a supervised clustering approach to estimate the quality of program module.

The proposed approach is based on constraint based clustering using Growing Neural Gas as underlying algorithm. During the Growing Neural Gas clustering process, the constraint maintains membership of modules to clusters that are already labeled as either fault prone (*fp*) or not fault prone (*nfp*). The proposed approach is supervised because the clustering process is aided with a set of labeled program modules. To our knowledge, this is the first study to investigate supervised clustering using Growing Neural Gas for the software quality modeling, analysis, and estimation problem.

Clustering is an appropriate choice for software quality analysis [6]. Given software measurements (attributes) of the labeled program modules, clustering algorithms group the modules according to similarity of their software attributes. The underlying assumption is that program modules with similar attributes will have similar quality characteristics. Hence, *fp* modules will have similar software measurements and will likely group together as clusters. Similarly, *nfp* modules will also group together as clusters. We investigate the proposed approach using software measurement and defect data from a previously developed National Aeronautics and Space Administration (NASA) software project. The end result after a run of the proposed supervised clustering approach is a set of labeled clusters. The labeled clusters are the result of the labeling during clustering process from training data set PC1.

In context to software quality estimation, most research have focused on clustering using K-means, Mixture-of-Guassians, Self-Organizing Map, Neural Gas[7][8][9][11]. In all these techniques we are required a predefined structure that is numbers of neurons or clusters before we start clustering process. To avoid the need of pre-determining the quantity of neurons and the topology of the structure to be used, other clustering techniques start with a minimal neurons structure that is incremented during training until it reaches a maximum number (or a optimal number of neurons, in some sense), or until the network reaches a minimal error regarding data quantization. Examples of these algorithms include the

Dr. Parvinder S. Sandhu is Professor with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 INDIA (Phone: +91-98555-32004; (Email: parvinder.sandhu@gmail.com)

Sandeep Khimta is Lecturer with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-India

Kiranpreet Kaur is doing her Masters in Computer Science & Engineering from Rayat Institute of Engineering & Information Technology, Rail Majra, Distt. Distt. Nawanshahr ,144533, Punjab.

Growing Cell Structures (GCS) [10] and the Growing Neural Gas [11], both developed by B. Fritzke.

The remainder of this paper continues with Section II, where a brief discussion on working of Growing Neural Gas clustering algorithm is presented. The proposed software quality analysis using Growing Neural Gas as underling clustering scheme is detailed in Section III, followed by Section IV, in which result and discussion is presented on methods used to analyze and in section V, conclusion is presented.

II. Growing Neural Gas

Given any input data distribution in R^n, GNG incrementally creates a graph in which each vertex corresponds to a unit (neuron) formed by a reference vector w_i in R^n $(w_i = [w_{i1}, w_{i2} ..., w_{in}])$, an $error_i$ variable representing accumulated local error and an edge group defining unit i topological neighbourhood. Reference vector represents unit position in output space. At each iteration a new input signal x is analyzed. Two iteration winner nodes are chosen, s and t, such that $|w_s - x|$ is the smallest value and $|w_t - x|$ is the second smallest value, where $|\ |$ represents Euclidean distance. Winner node adjusts its accumulated local error, adding Euclidean distance between unit and analyzed input vector, according to equation below [10][11]:

$$\text{errors} \leftarrow \text{errors} + |\, w_s - x \,| \quad (1)$$

Accumulated local error represents the measure used to define new units' insertion place. According with Competitive Hebbian Learning (CHL), a connection with age 0 is established between winner unit and second place. The winner unit and its neighbors, established by CHL, are moved a fraction toward the input data's direction, according with following equations [10][11]:

$$w_s \leftarrow w_s + e_b(x - w_s) \quad (2)$$

$$w_n \leftarrow w_n + e_n(x - w_n) \quad (3)$$

$\forall\, n \in$ Neighbor(s), where e_b is winner node updating fraction and e_n is neighbor updating fraction, e_b and $e_n \in [0,1]$. Here w_s is the reference vector of winner node and w_n is the reference vector of neighboring nodes of winner node. This causes GNG's training dynamics to keep input data topological relation. In the first iteration only two units are created, with its reference vectors randomly established. The algorithm is said growing, because each i iterations a new neuron is inserted. The insertion of a new unit occurs between the unit with highest (global) accumulated local error and its neighbouring unit with highest accumulated local error. This operation addresses a reduction of accumulated region error, and consequently, a global error reduction. Therefore, algorithm remains inserting new units until an established stop criterion is met. GNG performs also some edge pruning. A parameter, *amax,* is established, denoting maximum age that an edge can reach. In each iteration, ages of all graph edges are incremented. Edges with ages reaching established maximum value are removed from graph. This rule is used to prevent edges generated by occasional data and that should be removed during the algorithm' iterations. Thus, an aspect to be analyzed in defining an automatic data classification is the generation of different graph connected component by the algorithm. Each component would correspond to a different class. Data submitted to a trained map are associated to a winner unit and, consequently, to a graph component representing a class. Nevertheless, some factors affect negatively the GNG's formed graph. According with CHL policy used by algorithm, one input signal only is enough to generate a connection between two units. This causes outlier data, or extremity class data, to have great influence over generated graph connections. For further details of GNG refer to reference [10][11].

III. Proposed Work

The proposed clustering approach for software quality analysis is a constraint-based scheme that uses labelled instances from training data set for initial seeding (centroids) of clusters among the maximum allowable clusters—a user defined quantity when using Growing Neural Gas as the clustering algorithm. The proposed algorithm used in our constraint-based clustering approach with Growing Neural Gas is enumerated as follows.

Step1: Obtain the required set of clusters using Growing Neural Gas clustering algorithm from training data set with each cluster's centroid initialized.

Step2: Assign the class labels *fp* and *nfp* to the obtained set of clusters based on the criteria that if c_i is a cluster m_i is program module from training data set such that Euclidean distance between c_i and m_i is minimum, then assign the class labels *fp* or *nfp* of m_i to c_i.

Step3: Use the labeled set of clusters to predict the quality of test data set based on the criteria that if c_i is a cluster t_i is program module from testing data set such that Euclidean distance between c_i and t_i is minimum, then assign the class labels *fp* or *nfp* of c_i to m_i.

To predict the results, we have used confusion matrix. The confusion matrix has four categories: True positives (TP) are the modules correctly classified as faulty modules. False positives (FP) refer to fault-free modules incorrectly labeled as faulty. True negatives (TN) are the fault-free modules correctly labeled as such. False negatives (FN) refer to faulty modules incorrectly classified as fault-free modules.

TABLE I. A CONFUSION MATRIX OF PREDICTION OUTCOMES

		Real data	
Predicted		*Fault*	*No Fault*
	Fault	*TP*	*FP*
	No Fault	*FN*	*TN*

The following set of evaluation measures are being used to find the results:

- *Type I Error:* The value of Type I error can be calculated using following equation:

$$Type\ I\ Error = \frac{FP}{TN + FP} \quad (4)$$

- *Type II Error:* The value of Type II error can be calculated using following equation:

$$Type\ II\ Error = \frac{FN}{TP + FN} \quad (5)$$

If the value of Type I and Type II errors is low then the proposed system is more accurate in prediction of fault prone modules.

- *Overall Error:* The value of Overall error can be caluculated using following equation:

$$Overall\ Error = \frac{FN + FP}{TP + FN + FN + FP} \quad (6)$$

IV. Result and Discussion

The software measurements and quality data used in this paper to investigate the proposed work are those of a large NASA software project PC1 and CM1. The PC1 project is a flight software from earth orbiting satellite that is no longer operational. It consists of 40 KLOC code of C. The data were made available through the Metrics Data Program (MDP) at NASA (http://mdp.ivv.nasa.gov/) and included software measurement data and associated error (fault or defect) data collected at the function level. PC1 contain 1107 modules of which 76 contains one or more faults and 1031 contains zero faults. The maximum numbers of faults in a module is 9. The CM1 is a science instrument application written in C code with 20KLOC. The data set contain 505 modules out of which 48 contains one or more faults and 457 have zero faults. The maximum numbers of faults in modules is 5. In this paper, a program module with no faults was considered nfp and fp otherwise.

Each program module in the PC1 and CM1 was characterized by 21 software measurements [14]: 13 core metrics (as shown in Table II) and eight derived Halstead metrics (Halstead_Length, Halstead_Volume, Halstead_Level, Halstead_Difficulty, Halstead_Content, Halstead_Effort, Halstead_Error_Est, and Halstead_Prog_Time). In this study only 13 basic software metrics are used. The eight derived Halstead metrics were not used as they are derived from basic Halstead metrics.

TABLE II. Software Measurements In PC1 And CM1 Data

Line count metrics	LOC_BLANK LOC_CODE_AND_COMMENT LOC_COMMENTS LOC_EXECUTABLE LOC_TOTAL
MaCabe metrics	CYCLOMATIC_COMPLEXITY DESIGN_COMPLEXITY ESSENTIAL_COMPLEXITY
Helstead metrics	NUM_OPERANDS NUM_OPERATORS NUM_UNIQUE_OPERANDS NUM_UNIQUE_OPERATORS
Branch count metric	BRANCH_COUNT

TABLE III. Notation Used For Result Analysis

Symbol	Description
n	total number of modules
nfp	not fault prone module
fP	fault prone module
P_nfp	predicted not fault prone module
P_fp	predicted fault prone module
TYPE I error	nfp modules predicted as fp
TYPE II error	fp modules predicted as nfp
Overall	total misclassification rate

Table III shows the different symbols and their description that are used in our result analysis part.

The initial numbers of clusters and their label *nfp* and *fp* clusters were determined based on the algorithm presented in Section IV. We have implemented proposed algorithms described in section III in Matlab 7.4 environment.

TABLE IV. Initial Set Of Clusters For Different Values Variable Lembda

S.N.	λ	Total number of clusters
1	50	25
2	70	18
3	90	15
4	110	13
5	130	11
6	150	10

PC1is used as training data set for the proposed work and Table IV shows the total numbers of cluster sets that is achieved in step1 of the proposed algorithm for different values of λ. The λ is the number of iterations after which new cluster is created in GNG algorithm.

For the given software measurement data set, if the above described classification was performed several times, the expected Type I and Type II error rates would be governed by the proportions of the two classes, i.e. *nfp* and *fp*. If a data set consists of 900 *nfp* and 100 *fp* modules, a randomly selected module has a 0.10 probability of being correctly predicted as *fp* (p = 10%). Similarly, a module has a 0.90 probability of being correctly predicted as *nfp* ($1 - p = 90\%$). Assuming that 200 modules are randomly predicted as *fp* and the remaining 800 modules are predicted as *nfp*, then the expected Type I and Type II error rates are p and $(1 - p)$, respectively: Type I = $(0.10 \times 800/800)$ and Type II = $(0.90 \times 200/200)$. Among the 200 modules predicted as *fp*, the expected number of correct predictions is $0.10 \times 200 = 20$, and among the 800 modules predicted as *nfp*, the expected number of correct predictions is $0.90 \times 800 = 720$.

TABLE V. TEST DATA MISCLASSIFICATION RATE WITH PROPOSED WORK AND CM1 AS DATA SET

λ	Test Set No.	*Type I (in %)*	*Type II (in %)*	*Overall (in %)*
λ =50	1	36	95	41
	2	89	33	84
	3	3	85	11
	4	25	39	27
λ =70	1	4	85	11
	2	76	33	71
	3	26	42	27
	4	87	10	80
λ =90	1	78	46	75
	2	23	46	26
	3	10	75	16
	4	25	42	26
λ =110	1	56	62	57
	2	56	6	52
	3	25	43	26
	4	20	56	23
λ =130	1	79	33	75
	2	22	48	22
	3	17	56	20
	4	21	56	22
λ =150	1	72	31	68
	2	35	31	34
	3	76	23	71
	4	43	15	40

TABLE VI. BEST CASES OF TEST DATA MISCLASSIFICATION RATE WITH PROPOSED WORK FOR CM1 DATA SET

λ	*Type I*	*Type II*	*Overall*
50	25	39	27
70	26	42	27
90	25	42	26
110	20	56	23
130	22	48	22
150	35	31	34

In the present work CM1 is used as test data set. Type I, Type II and overall error for different values of λ was calculated. For each value of λ four different sets are conducted to get better insight of prediction result of proposed work. The expected classification results are summarized in Table V for different values of **λ.** In table VI best predicted result for each value of λ are recorded. It is clear form the observation of table V that as the value of λ increases the value of Overall error rate decreases and the value of Type II error increases. When λ is set to 150 we get exceptional values that is: Overall error rate increases and the value of Type II error decreases. Lowest Overall error rate is observed when λ is set to 130; where as lowest Type II error rate is observed when λ is set to 150. The prediction result is considered best if the Overall error rate and Type II error rate are on lower side. Here the best result is achieved for when λ is set to 150. The best combination is when λ is set to 150 where we get lower value combination for both Type II and Overall error rate.

V. CONCLUSION

Software quality assurance plays important part of any software project. The prediction of Software quality during development life cycle of software project helps the development organization to make efficient use of available resource to produce the product of highest quality. In the proposed work Growing Neural Gas clustering technique is used to predict the quality of software project. The output of proposed model is prediction of module as faulty one or not faulty. The performance of the proposed model is measured in terms of Type I, Type II and Overall error rate. Lower the value of overall and Type II error rate better is the quality prediction rate of that model. The best testing results shows 80% accuracy in evaluating the quality of software modules. Hence, the proposed technique can be used by programmers in evaluating the quality of modules during software development.

REFERENCES

[1] K. E. Imam, S. Benlarbi, N. Goel, and S. N. Rai, "Comparing casebased reasoning classifiers for predicting high-risk software components",J. Syst. Softw., vol. 55, no. 3, pp. 301–320, Jan. 2001.

[2] N. F. Schneidewind, "Investigation of logistic regression as a discriminant of software quality," in Proc. 7th Int. Softw. Metrics Symp., London, U.K., Apr. 2001, pp. 328–337.

[3] Taghi M. Khoshgoftaar, Naeem Seliya, "Analogy-based practical classification rules for software quality estimation," Empir. Softw. Eng. J., vol. 8, no. 4, pp. 325–350, Dec. 2003.

[4] L. Guo, B. Cukic, and H. Singh, "Predicting fault prone modules by the Dempster–Shafer belief networks," in Proc. 18th Int. Conf. Automated Softw. Eng., Montreal, QC, Canada, Oct. 2003, pp. 249–252.

[5] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," Empir. Softw. Eng. J., vol. 9, no. 3, pp. 229–257, Sep. 2004.

[6] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques," IEEE Intell. Syst., vol. 19, no. 2, pp. 20–27, Mar./Apr. 2004.

[7] T. Kohonen, Self-Organizing Maps, 3nd. Ed., Berlim: Springer. 2001.

[8] S. Goldman and Y. Zhou, "Enhancing supervised learning with unlabeled data," in Proc. 17th Int. Conf. Mach. Learn., Jun. 2000, pp. 327–334.M. Young, *The Techincal Writers Handbook.* Mill Valley, CA: University Science, 1989.

[9] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," Empir. Softw. Eng. J., vol. 9, no. 3, pp. 229–257, Sep. 2004. R. W. Lucky, "Automatic equalization for digital communication," *Bell Syst. Tech. J.*, vol. 44, no. 4, pp. 547–588, Apr. 1965.

[10] B. Fritzke. Growing Cell Structures - A Self-organizing Network for Unsupervised and Supervised Learning. Neural Networks, 7(9):1441-1460, 1994.

[11] B. Fritzke, "A Growing Neural Gas Network Learns Topologies." Advances in Neural Information Processing Systems, pp. 625-632, 1995.

[12] K. A. J. Doherty, R. G. Adams, N. Davey. "Hierarchical Growing Neural Gas." Int. Conf. on Adaptive and Natural Computing Algorithms, 2005.

[13] Taghi M. Khoshgoftaar, Naeem Seliya, "Software quality classification modeling using the SPRINT decision tree algorithm", In Fourth IEEE international conference on tools with artificial intelligence (pp. 365–374). 2002.

[14] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2nd ed. Boston, MA: PWS-Kent, 1997.